Extending the Representational Power of Model-Based Systems using Generalized Timelines

Russell Knight, Gregg Rabideau, Steve Chien
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Keywords: Scheduling, Representation, Planning, Resources, Model Based Systems

Abstract

Current declarative systems (i.e., model-based systems) are limited to a small number of types of states and resources that they can represent (e.g., we cannot model orientation with current systems). Thus, software is usually "hand-crafted" to meet the needs of state validation-estimation-projection for real domains where estimation is not sufficient. To meet this need, we provide an abstract formulation of capacitated resource scheduling in metric time that admits a straightforward procedural interface for easy customization meeting many real-world domain description requirements while remaining in the same complexity class as their simpler scheduling counterparts—NP-Complete.

1 Introduction

Current declarative systems that model states and resources have been fairly successful, yet many domains lie out of the reach of these systems due to a lack of representation sufficiency. Thus, software is usually "hand-crafted" to meet the needs of state and resource validation-estimation-projection. Although existing systems are usually sufficient to approximate resource and state usage, in many cases approximations are not sufficient due to the high cost of "false positives" for some domains. For example, many NASA missions are extremely risk averse and require very small margins of error. Many unique systems require specialized representations (e.g., conductivity ratings at various temperatures for conductors), and the resources are not available to develop a declarative system for each new type of technology that we need to model. For these reasons, it is not clear that the procedural "customized" aspect of modeling real domains will ever vanish.

To address this dilemma, we provide a formulation of generic validation-estimation-projection modeling constraints along with the procedural interface to them. To achieve this formulation, we have surveyed many real-world domains and characterized their similarities and differences. Given this formulation, we provide algorithms for reasoning in a generic way about state validation, estimation and projection. Our hope is that this formulation becomes somewhat standard, and other researchers will extend and improve these algorithms.

2 Semantic Decomposition

Semantic decomposition is a process that reduces a set of operators and operands for a collection of domains to a common set of abstract operators and operands. For example, consider the domains of "summing integers" and "multiplying real numbers". Each of these domains consists of operators (+ and ·) and operands (integers and real numbers). Now, the semantics of the + operator with respect to integers is not the same as the semantics of the · operator with respect to real numbers. But, consider the relationship of the + operator and the · to their operands. We see that they have certain properties in common, e.g., commutativity ($1+2 = 2+1$, $1.1 \cdot 2.1 = 2.1 \cdot 1.1$), associativity ($1+(2+3) = (1+2)+3$, $1.1 \cdot (2.1 \cdot 3.1) = (1.1 \cdot 2.1) \cdot 3.1$), and identity ($2+0 = 2$, $2.1 \cdot 1 = 2.1$). Thus, we abstract away the semantic content and keep the "relationship" content of the operators, operands, and some operand values (e.g., the identity value). Thus, we can talk about abstract operators and abstract operands and abstract values for operands.

Extending our example, we create an abstract operator **M** that we call *merge*, a set of abstract operands O and an abstract operand value $\varnothing \in O$. This is useful in that we can produce a system for reasoning about both the addition of integers and the multiplication of real numbers. For example, if we have an operand value o from the set of all possible operand values O, identity is expressed as $o\mathbf{M}\varnothing = o$. Of course, our goal is to model domains with more operators and relationships than these.

Note that this has strong ties to the algebraic concepts addressed by *group theory*, although the algebras we produce are not groups per se because we do not require the *inverse* relationship to hold. This is because our goal is to model constraint systems, and an inverse relationship would imply that certain constraints exist that relax other constraints—a relationship that does not hold for the majority of the domains we have sampled.

Semantic decomposition in the context of validation-estimation-projection consists of 4 steps:

1. Formulate each domain in terms of a value over time and a set of constraints on that value.
2. Identify the operators and operands of the constraints.
3. Identify the interactions between operators and operands implied by the constraints.
4. Reformulate commonalties as aspects of a generic validation-estimation-projection (GVEP) problem and differences as aspects of a procedural representation.

For example, we have applied this analysis to existing representational systems. After all, we do not wish to diminish current representational capabilities. In general, current declarative reasoning systems are capable of modeling three domain types (a domain type is a domain that can represent many domain instances):

1. Depletable Resource – e.g., a battery that looses its charge when used and requires explicit replenishment. Constraints for this domain are the minimum and maximum allowed value and usage effects.
2. Non-depeletable Resource – e.g., a bucket that looses capacity when in use but immediately regains the capacity when not in use. Constraints for this domain are the minimum and maximum allowed value, and usage requirements.
3. Discrete Symbolic State – e.g., a stoplight that is in a discrete state (*red*, *yellow*, or *green*). Constraints for this domain are value transition constraints (e.g., for a stoplight, a *green* state must be followed by another *green* state or a *yellow* state but not a *red* state), and usage requirements (e.g., the light must be *green*), and delta effects (e.g., an ambulance causes the light to change to *green* by sending it a radio signal).

Now, let us consider the similarities and differences between these domains and constraints. The first step of semantic decomposition (with respect to validation-estimation-projection) is to divide the system in question into values over time and constraints on those values. In this case, the values over time are discrete real values and arbitrary symbols. The constraints are minimum values, maximum values, default values, non-depletable usage-requirement constraints that occur over a duration, depletable usage-effect constraints that occur at a moment and propagate forward in time, symbolic transition constraints, symbolic constraints that absolutely change a value and propagate it forward in time, and symbolic equality constraints that require the symbol to be equivalent to the constrained value over an interval.

The second step of semantic decomposition is to identify operators and operands of the constraints and values. The sum operator (e.g., usage requirement constraint) and the "less than" operator (e.g., maximum value constraint) are those used for real values. The equality operator (e.g., equality constraint) and assignment operator (e.g., delta constraint) are those used for symbols. Operands are real values and symbols.

The third step of semantic decomposition is to identify the relationships between the operands and operators. Here we see a similarity between states and resources. Non-depletable constraints affect the value over a specified temporal extent—so do symbolic equality constraints. Depletable constraints affect the value from now until the future—as do symbolic delta constraints.

The fourth step is to reformulate the constraints and values as a GVEP problem. In essence, the relationships between operands and operators, as well as the similarity between these entities are what define the GVEP problem solver. In this case, we see the requirement to propagate effects of constraints forward in time (as for state delta and depletable resource constraints), the requirement to reason about constraints whose effect is temporally contained by an interval (as for state equality and non-depletable resource constraints), and the requirement to reason about global constraints (state transition constraints and minimum/maximum value resource constraints). We note that global constraints are instances of constraints whose effect is temporally contained by an interval; thus, we combine these two types of constraints. To interface to the GVEP problem solver, we must provide the semantic description of the relationships between the operators and operands of the specific domain to be modeled. This is modeled procedurally.

A surprisingly small number of operators are sufficient to model a variable over time in the context of our current semantic decomposition (for depletable and non-depletable resources, and symbolic states). In terms of operands, we require a single operand type, O. In terms of operators, we require:

1. A merge operator ($x\mathbf{M}y$, $x \in O$, $y \in O$) that is associative and commutative. We use this to merge operand values together.
2. A value propagation operator ($x\mathbf{D}y$, $x \in O$, $y \in O$) used for calculating the "down-stream" effects of a value x on the value y given the context of the current value y. This is used to compute values after delta constraints.
3. A consistency checking predicate $\mathbf{OK}(x)$ that is used to validate that a particular operand value is not violating any constraints associated with it.

4. A consistency checking predicate **XOK**(x,y) that is used to validate that a transition from the first operand value (x) to the second (y) is not a violation of any constraints.

We have demonstrated that these are the necessary and sufficient requirements on the procedural semantic description for our GVEP problem solver to work for depletable and non-depletable resources, and symbolic states.

These three domain types can represent surprisingly many specific domains. Some domains, however, lie outside the representational sufficiency of these domain types. For example, consider the problem of modeling a file system. We wish to model aspects such as total memory capacity, the requirement for the existence of files, and the creation/modification/deletion of files. A depletable resource could model memory use, but a symbolic state is insufficient to model file existence requirements. Normally, including such a new type of domain in a declarative system would require custom code for both the semantic representation and the reasoning algorithms. Our approach removes the requirement for custom coding the reasoning algorithms and reduces the overhead required to encode the semantic representation. Interestingly, we find that the current semantic decomposition suffices.

3 GVEP

We address the following issues with respect to the GVEP problem (we discuss the problem solver later):

1. How do we project the effect of constraints and the current state of the system over time with respect to future constraints? Likewise, how do we estimate future and current values without actual measurements? These questions are answered in the sections concerning Evaluating the Assignments to the Operand Values over Time, Effects on Temporal Extent, Computing Operand Values, and Consistency.
2. How do we validate that no constraints are violated or, if constraints are violated, how do we identify the violated constraints? This question is answered in the sections concerning Consistency, and the Problem Solver.
3. What algorithms can we use to introduce more constraints without causing violations or to remove constraints to alleviate violations?

The key to this approach is that we expect the semantics of a variable to be described by the user or a developer, and we provide the reasoning engine. Thus, with the definition of a few functions, a developer has all the power of a complete, declarative validation-estimation-projection reasoning system.

Another key aspect of this approach is that we model a series of discrete operand values over time. We call such a series of operand values a *timeline*. The operand value is a combination of the actual projected value and the constraints on the actual value. For example, a symbolic state might be *green* from time t to time t+1. (Note that times need not be grounded—they might just as well be time-points in a temporal network representing flexibility, assuming that the time-points are ordered.) If there were a constraint that the symbolic state be *green* over the same time, then its operand value would reflect both the actual value (it is *green*) and the value of the constraint (it must be *green* to be valid).

Evaluating the Assignments to the Operand Values over Time

Given a procedural semantic description of a domain, and a set of instances of constraints for the domain, we wish to represent the discrete operand values implied by the constraints over time. In other words, we wish to compute the timeline. What follows is a description of exactly how the timeline is generated. Given a timeline, we can determine which (if any) constraints are violated and perform other reasoning tasks associated with providing the functionality of the GVEP problem solver.

Consider a series of operand values V. Initially, only a single operand value v exists in V and it spans the entire timeline. v is assigned a value of $\varnothing$, which is a universal value for all variables. With no other constraints on v, it would remain such. A constraint on a timeline represents a merge operation of the current value of the timeline and the value of the constraint over a specified period of time or temporal extent. Note that this disallows default values, but default values are subsumed under generic constraints since we can place a constraint at the very start of a timeline. We model the timeline as a series of discrete temporal segments that completely cover the horizon yet do not overlap. Note that the temporal divisions between operand values over time need not be definite assigned values but could be time points in a temporal network, making this system extendable to such current and proposed systems as the Remote Agent and the Mission Data System.

All constraints have a start- and end-time point. The temporal extent of a constraint is its start-time inclusively up to its end-time non-inclusively. Note that this is not necessarily the temporal extent of the effects of the constraint because constraints also may have propagation properties. Consider the timeline v in Figure 1 with two constraints that have no unusual effects. We see the timeline as a series of operand values and the constraints as a collection of operand values and temporal extents. In this example, α and β represent specific operand values.

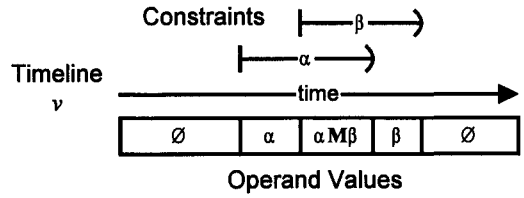

Figure 1 Simple generic timeline

Effects on Temporal Extent

We define one more attribute of a constraint: its *effect*. The effect of a constraint determines any special consideration it should be given in terms of its temporal extent. Currently, we define only two effects: 1) *local* and 2) *downstream*. A constraint with a local effect only influences assignments of the timeline it constrains within its temporal extent, as in Figure 1. If the effect of a constraint extends beyond its defined temporal extent, we say that its value affects the timeline downstream. Consider Figure 2, which is the same as Figure 1 except that its constraints' effects are downstream.

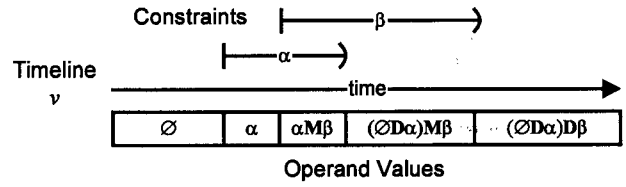

Figure 2 Abstract timeline with constraints of both local and downstream effect

For example, consider a symbolic state. The effect of changing to a state (e.g., *green*) in the schedule propagates forward in time until something occurs to cause another change (e.g., an ambulance or the mechanism of the traffic light). In this sense, we can extend the temporal extent of a constraint by endowing it with a *downstream* effect. But, we do not wish downstream values to always propagate down the entire timeline, e.g., after a traffic light changes, the previous values have no effect on the current value. Similarly, we might receive updates from measuring equipment (e.g., our eyes might tell us that the light is in fact *red*). Clearly, we do not want previous constraints' effects to propagate beyond an absolute measurement.

It is important to note that more effects are possible. Consider a constraint that propagates back in time instead of forward, or a constraint that causes no splits in timeline segments, i.e., it simply merges with the existing segments, possibly extending its temporal extent. These are feasible, but not necessarily reasonable, and are not included in our discussion.

Computing Operand Values

To compute the operand values of a generic timeline, we use three timelines: the local timeline, the downstream timeline, and the visible timeline. The operand values that make up the visible timeline are the result of merging the operand values of the local and downstream timelines. Let us use the previous example in Figure 2 but now show the internal workings of the timelines (Figure 3; keep in mind that $x\mathbf{M}\varnothing = x$).

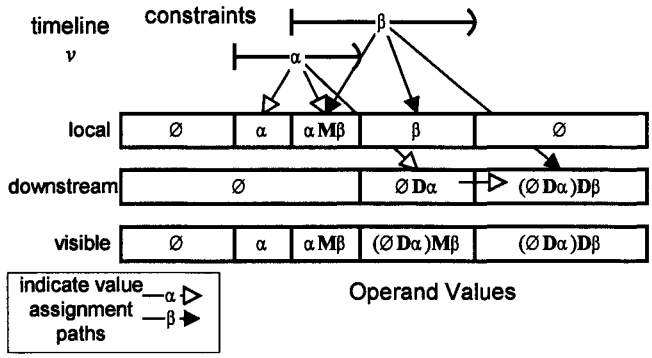

Figure 3 Internal workings of the timeline

Here we see that the downstream assignments are a series of **D** operations using the operand value of the constraint and applying it at the end of the constraint. We use the function **D** to combine values that are most recently contributing to the series with the preceding operand value. We do this to provide an overloading capability for the most recent information. This concludes our discussion of timeline computation. We continue with a discussion on operand value consistency validation.

Consistency

Checking consistency of a timeline is a two-phase operation: 1) check the consistency of each visible operand value and 2) check the consistency of each transition. Given **OK** and **XOK**, consistency can easily be validated in time that is proportional to the number of operand values in the visible timeline for a given timeline. The number of assignments for the timeline is proportional to the number of constraints constraining the timeline, so consistency checking for a timeline is roughly proportional to the number of constraints constraining the timeline. Of course, one could implement an incremental update to the consistency status as one updates values on the visible timeline. Transitions can be validated using the same technique.

File System Procedural Semantic Description Example

We now explore our example in detail. We wish to define a domain that models a file system. Specifically, we wish to represent constraints on the total amount of space available, the amount of memory used by individual files, and the requirement of file availability. We also need to model normal file operations such as creation, modification, and deletion with respect to our earlier constraints. We formulate a file system as a GVEP problem by providing a minimal set of definitions.

An operand value v for this domain would consist of the following:

1. A measurement of the total space (used or not) of the system
2. A set of files and space allocations
3. A set of requirements that imply that a file must exist

4. A set of pending deletions

The identity value ∅ would consist of the following:

1. Total space would be infinity, as this is the least constraining value
2. An empty set of files
3. An empty set of required files
4. An empty set of pending deletions

The merge operator **M** would behave thusly:

1. Take the minimum of the total space values as the resultant total space value.
2. Take the summed union of the sets of files. That is, if a file exists in both, it exists in the resultant set with the usage being the sum of usages. If a file exists in only one of the two sets of files, it exists in the resultant set with the same usage.
3. Take the union of the sets of required files.
4. Take the union of the sets of pending deletions.

The downstream propagation operator **D** would behave thusly (note left-hand operand is *previous*, right-hand operand is *current*):

1. Take the minimum of the total space values as the resultant total space value.
2. Remove all files in the file list of the previous operand that are in the deletion file list of the previous operand. Take the summed union of this file set with the current file set.
3. Take the union of the sets of required files.
4. Use the current deletion file set only.

The operand value consistency-checking predicate **OK** is defined as such: If the sum of all usages of files of the file list that have no matching file in the deletion list is greater than the total space, return false. Otherwise, if any file exists in required file list that doesn't exist in the file list, return false. Otherwise, everything is ok, so return true.

The operand transition consistency-checking predicate **XOK** always returns true since any transition is allowed.

These examples demonstrate that the GVEP timeline representation is sufficient for current declarative systems, as well as domains that current systems cannot represent. The proposed research will take this system farther than current systems and provide a framework upon which the procedural description of a domain can take advantage of a declarative validation-estimation-projection system with low overhead.

Problem Solver

Until now, we have focused on representation. We have assumed a solver in the background that given the procedural semantic description of a domain can solve validation-estimation-projection problems. We must address the details of the algorithm and, therefore, we now turn our attention to the issue of solving specific validation-estimation-projection problems. The only assumption we make about the procedural semantic description of a domain is that the asymptotic space and time complexity of each operand value scales at worst linearly with the number of operations (merge or downstream propagation) performed on it.

Generic Constraint Validation—we now turn to validating a set of constraints given a procedural semantic description. By constraint, we mean an object that has a start time point, an end time point, an operand value, and an effect, as described earlier. The time points are associated to each other using *simple temporal constraints* [Dechter *et al.*, 1991]. These are usually in the form of "time point *a* must come after time point *b* by at least 10 minutes but not more than 15 minutes." The question we would like to answer is this: Is there any assignment to the time points of the constraints such that the resulting timeline is consistent (as described earlier using **OK** and **XOK**)? Or, conversely, is it true that any assignment to the time points is ok? As it turns out, these are very difficult questions. Considering that we wish to ask about validity often, we must make a compromise on the types of validations we perform. Our compromise is that we only validate collections of constraints that have time points that are totally ordered with respect to any given domain.

More formally, generic constraint validation is as follows:

Generic Constraint Validation (GCV)

Given a simple temporal problem [Dechter *et al.*, 1991] $G=(V,E)$, $w(e\in E)\rightarrow Z$, a set of domains S, a set of constraints C, $c=(tp_1\in E,\ tp_2\in E,\ f\in\{local,\ downstream\},\ s\in S)\in C$.

QUESTION: Is there a legal assignment to the time points such that the constraints are not violated?

Of course, we often are concerned with the negation of this. That is to say, is there a legal assignment to the time points such that the variable constraints are violated? One example of a scheduler (and planner) that handles this problem at its most general level (but only for real valued resources) is the IxTet system [Laborie and Ghallab, 1995 and Vidal and Regnier, 1999]. IxTet uses a maximal clique approach to discovering if a resource has been over-subscribed. Unfortunately, computing a maximal clique is NP-complete. Thus, we wish to impose certain bounds on the types of problems that are validated. If all time points associated with a variable through a constraint are totally ordered, then validation becomes trivial because only one series of values for a variable is possible (albeit with varying times associated to the divisions between the values). We adopt this approach to validation—that is, we assume that all time points are totally ordered with respect to a domain before validating them (checking for constraint violations using the **OK** and **XOK** predicates for each variable). This approach is similar to previous approaches [Cesta and Smith, 1998 and Muscettola *et al.*, 1998].

Thus, the remaining problem is this: Given a collection of constraints on a domain or domains related to each other via temporal constraints between their time points, is there a

way to force a total ordering among constraints of common domains such that the resultant timeline is consistent? We see this as a precedence constraint problem, where the idea is to introduce temporal constraints that impose an ordering on the time points of the domain constraints. This technique is referred to as precedence constraint posting and has been used effectively for solving scheduling problems [Cesta and Smith, 1998].

Generic Constraint Ordering Problem (GCOP)

Given a simple temporal problem [Dechter *et al.*, 1991] $G = (V,E)$, $w(e \in E) \rightarrow Z$, a set of variables S, a set of constraints C, $c = (tp_1 \in E, tp_2 \in E, f \in \{local, downstream\}, s \in S) \in C$.

QUESTION: Is there a set of precedence constraints P that can be added to the existing temporal constraints such that a total ordering is imposed on all points associated with a common variable $s \in S$? A precedence constraint is of one of two forms: 1) time point x strictly precedes time point y by ε where ε is an arbitrarily small amount of time, or 2) time point x is simultaneous to time point y. In the latter case, we simply collapse time points x and y to be the same time point.

To apply the appropriate algorithm to solve this problem, we would like to know just how difficult it is. We have proved that this problem is NP-complete.

Knowing that GCOP is NP-complete, we choose to apply a search technique that has many favorable characteristics for NP-complete problems: Depth First Branch and Bound (DFBnB). DFBnB has the following traits: It can return a solution once it has searched to depth (in this case, the number of constraints necessary to induce a total ordering), thus making it an any-time algorithm with respect to the GCOP.

- It has been used successfully in solving optimization problems faster than any other known technique [Zhang, 2000].
- It can make use of heuristic information to aid it in finding solutions faster.
- It will find the optimal solution.
- It will terminate.

The first step in applying DFBnB to a problem is to define the search space. In the case of GCOP we are searching the space of precedence constraint assignments.

The next step in applying DFBnB is to develop heuristics for estimating the cost of a solution given our cost and solution context. The development of heuristics is a major area of research, and due to space constraints we omit such a discussion. It is important to note that heuristics must be admissible for DFBnB to find optimal solutions.

4 Future Work

There is an obvious underlying planning problem that deals with issues such as when to insert new constraints and what value to assign them. Future work should address these issues.

5 Acknowledgements

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[Cesta and Smith, 1998] Oddi S. Cesta and S. Smith. "Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems." *Proc. AIPS98*, pp. 214-223, 1998.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl "Temporal Constraint Networks," *Artificial Intelligence*, 49, 1991, pp. 61-95.

[Fukunaga *et al.*, 1997] A. Fukunaga, G. Rabideau, S. Chien, D. Yan. "Towards an Application Framework for Automated Planning and Scheduling," *Proc. of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space*, Tokyo, Japan, July 1997.

[Karp, 1972] R. M. Karp. "Reducibility among combinatorial problems." In R. E. Miller and J. W. Thatcher (eds.) *Complexity of Computer Computations*, Plenum Press: New York,pp. 85-103, 1972.

[Knight *et al.*, 2000a] R. Knight, S. Chien, T. Starbird, K. Gostelow, and R. Keller. "Integrating Model-based Artificial Intelligence Planning with Procedural Elaboration for Onboard Spacecraft Autonomy." *SpaceOps 2000*, Toulouse, France, June 2000.

[Knight *et al.*, 2000b] R. Knight, G. Rabideau, and S. Chien. "Computing Valid Intervals for Collections of Activities with Shared States and Resources," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 14-17 April 2000, pp. 339-346.

[Laborie and Ghallab, 1995] P. Laborie and M. Ghallab. "Planning with Sharable Resource Constraints," Proc. IJCAI-95,pp. 1643-1649, 1995.

[Muscettola *et al.*, 1998] N. Muscettola, P. Nayak, B. Pell, and B. Williams. "Remote Agent: To Boldly Go Where No AI System Has Gone Before," Artificial Intelligence 103(1-2):5-48, August 1998.

[Vidal and Regnier, 1999] T. Vidal and P. Regnier. "Total Order Planning Is More Efficient Than We Thought." *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, AAAI Press, 1999, pp. 591-596.

[Zhang, 2000] W. Zhang. "Depth-First Branch-and-Bound versus Local Search: A Case Study." *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, AAAI Press, 2000, pp. 930-935